Apple IIGS
#91: The Wonderful World of Universal Access

Revised by: Dave Lyons May 1992
Written by: Don J. Brady, Matt Deatherage, & Ron Lichty September 1990

This Technical Note discusses how your applications can be compatible with Universal Access software.
Changes since July 1991: Added caution against reading the keyboard with interrupts disabled.

What's "Universal Access?"

Universal Access is the name given to software components designed to make Apple computers (in this case, the Apple IIGS) more accessible to people who might have difficulty using them. The Apple IIGS is very dependent on graphic objects, a keyboard and mouse; not all people can use these things very easily.

There are several components to Apple's Universal Access software:

- **CloseView**. CloseView magnifies the Apple IIGS screen so that it's more easily seen by those with visual impairments. The hardware screen contains a magnification from two to twelve times as large as the "real" 32K Super Hi-Res graphics screen.

- **Video Keyboard**. Video Keyboard is a New Desk Accessory that emulates a keyboard. A picture of a keyboard appears on the screen; a mouse-down event in any "key" makes Video Keyboard post a key-down event, so you can use a pointing device as a keyboard. ADB hardware is available to allow people to use head gear or other devices instead of mice; Video Keyboard lets these same devices replace the keyboard as well.

- **Easy Access.** Easy Access comes in two parts: Sticky Keys and Mouse Keys. **Sticky Keys** makes the keyboard easier to use for those who have trouble pressing more than one key at a time; while Sticky Keys is activated, modifier keys may be released and still apply to the next keystroke. **Mouse Keys** allows the numeric keypad to be used as a mouse substitute. Sticky Keys and Mouse Keys are included in all ROM 03 Apple IIGS computers. The software versions allow all Apple IIGS computers to provide these functions, and provide additional icon feedback (in the upper right menu bar) for Sticky Keys.

How It Works (Access Nothing and Checks For Free)

Universal Access generally works by replacing Apple IIGS toolbox functions. For example, CloseView patches QuickDraw so you do not draw to the actual screen, but to another buffer that CloseView can then magnify. Video Keyboard patches the Window Manager so that its keyboard window is always frontmost and fully visible (and accessible). Easy Access uses the ADB tools and the Event Manager to alter the way the hardware responds.

Since Universal Access changes the way the tools behave, your applications do not have to work very hard to be accessible to a broad range of physically challenged people. Just by following the rules, you have an accessible application. There are, however, a few guidelines you should keep in mind when designing your programs to make them as accessible as they can be.

Universal Access Compatibility Guidelines

- Don't disable interrupts and then try to read the keyboard. Easy Access on ROM 1 works at the Apple Desktop Bus level—if ADB interrupts are not being serviced, no keypresses will show up at $C000/$C025. Even Reset will not work, so the user may have to power down to regain control of the machine.

- Try to avoid using modal dialogs. Not only do lots of modal dialogs make for a cumbersome interface for everyone, they are especially annoying to those who have to move the mouse to a lot of OK buttons. More importantly, users cannot open NDAs like Video Keyboard while modal dialogs are frontmost.

  Video Keyboard can also be dragged in **front** of modal dialogs. If you are in the habit of using QuickDraw calls to draw items in Dialog Manager modal dialogs instead of creating custom dialog `userItems`, Video Keyboard users can drag the keyboard window in front of your dialog and erase the items (since the only items redrawn are those redrawn by the Dialog Manager's update routine). You can easily test this in all of your dialogs by obscuring each dialog with the Video Keyboard window a piece at a time, then moving Video Keyboard away, to be sure that all areas are completely redrawn.

  Let's say, for example, that you have a custom text item that changes between invocations of the same modal dialog. You might choose to draw the text yourself with `LETextBox2` after creating the dialog with `GetNewModalDialog` but before letting the Dialog Manager handle events with `ModalDialog`:

```
phx                         ; port: hi word from GetNewModalDialog
pha                         ; port: lo word from GetNewModalDialog
_SetPort

lda OurText+2               ; pointer to text to draw in modal dialog
pha
lda OurText
pha
lda OurTextLength           ; Text length
pha
```

```
        pea OurTextRect>>16         ; Text rectangle
        pea OurTextRect
        pea 0002                    ; Text justification (2 = fill)
        _LETextBox2
```

To be Universal Access–friendly, you would, instead, implement a `userItem` routine like the following:

```
; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
DrawDialogText
;
; DrawDialogText draws text pointed to by OurText into the Dialog.
; This userItem routine is called only by the Dialog Manager,
;   when it's implementing/updating the dialog.
; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

      lda >OurText+2             ; pointer to text to draw in modal dialog
      pha
      lda >OurText               ; (long addressing: data bank unknown)
      pha
      lda >OurTextLength         ; Text length
      pha
      pea OurTextRect>>16              ; Text rectangle
      pea OurTextRect
      pea 0002                   ; Text justification (2 = fill)
      _LETextBox2

      lda 1,s                    ; get return address
      sta 7,s                    ; move to proper location
      lda 2,s                    ;  above input parameters
      sta 8,s

      pla                        ; move stack pointer up
      pla                        ;  to new return address location
      pla
      rtl
```

It will be called as a result of adding a template item like the following to the dialog template (note use of Item Value for the text length, since template Value fields are not used by userItems):

```
   TextTemplate  dc.w 3                           ; ID
   OurTextRect   dc.w TTop,TLeft,TBottom,TRight
                 dc.w UserItem+ItemDisable        ; Type
                 dc.l DrawDialogText       ; Pointer to our userItem routine
   OurTextLength ds.w 1                           ; Text length (cheap place to put
it)
                 dc.w 0000                 ; Item flag
                 dc.l 00000000                    ; Item color
```

Note that this is a simple example of a custom item routine; if you really had custom text that changed from invocation to invocation, you could use the existing Dialog Manager `ParamText` and `longStatText2` item mechanisms.

- Use the Event Manager routines for event information. Do not access any hardware directly or use the lower-level Miscellaneous Tools routines for user event information—you steal that information from Universal Access. For example, use the Event Manager routine `GetMouse` to find the mouse location. Do not use `ReadMouse` or you steal mouse movement information from Universal Access.

- Call `GetNextEvent` or `TaskMaster` often. Long delays between calls do not let NDAs like Video Keyboard get events. If you cannot make these calls, at least call `SystemTask`.

- Do not assume that the hardware location of the screen is $E12000. Universal Access components that manipulate the entire screen (like CloseView) move the virtual screen so the hardware can be used for the magnified screen image.

  To find the screen location, look at the `ptrToPixImage` field in a `grafPort` after calling `OpenPort` (or in your window's window record after `NewWindow`). The image pointer gives the correct location of the screen.

  Assuming the current port is on screen, the following code finds the `ptrToPixImage` value:

```
pha
pha             ;made space for port pointer
_GetPort
phd             ;save direct page location
tsc
tcd             ;port pointer is now at 3..6 on direct page
ldy #4          ;offset to high word of ptrToPixImage
lda [3],y       ;got high word
tax             ;  in X
ldy #2          ;offset to low word of ptrToPixImage
lda [3],y       ;got low word
tay             ;  in Y
pld             ;restored direct page location
pla
pla             ;removed port pointer
```

  The `X` and `Y` registers now contain the base address of the screen.

- Do not assume things about being the frontmost window. Even if `FrontWindow` says you have the frontmost window, your `visRgn` may have pieces missing. For example, the title bar of your window may be partially under the menu bar. Or there may be a floating "windoid" (like Video Keyboard's window) over part of your window.

  For these reasons you should not draw directly to the screen without first examining your window's `visRgn`. Do not just check for rectangularity—your `visRgn` could be rectangular and parts of your window still be obscured. If you use QuickDraw for all your drawing, QuickDraw automatically clips drawing activity to be entirely within the `visRgn`, so this is not a problem.

- Don't access QuickDraw data directly; use QuickDraw routines instead. For example, to access SCB data, use the QuickDraw routines `GetSCB` and `SetSCB` instead of reading the hardware at $E19D00. CloseView may have those SCBs changed to reflect a magnified portion of the screen. Also use `GetColorEntry`, `SetColorEntry`, `GetColorTable`, and `SetColorTable`. Don't access the hardware directly.

- Try to allocate memory **after** starting the tools. If you want to allocate memory before starting tools, do not use special memory. (Set the `attrNoSpec` bit in the attributes.)

Further Reference

- *Apple IIGS Toolbox Reference*
- *Apple IIGS Firmware Reference*
- Apple II Video Overlay Card Development Kit (APDA)